

CSIT 5930 Search Engine and Applications

Homework 2

Released: Oct 13, 2020

Due: Nov 1, 2020, 11:59pm

In this homework, you build a search engine implementing the vector space model and tf*idf weighting method. You need to use Python to do the program and Flask to build the interface. If you are not familiar with Python and Flask, you should learn it (you won't regret learning them for your career). If you need more time to complete the homework, just drop me an email. I am flexible with the due date of this homework.

Install packages

- python 3.6+
 - Recommend using python3.6+
 - If you've never installed Python before, I would recommend installing "[Anaconda](#)". It is a toolkit that equips you with thousands of open-source packages and libraries.
- NLTK 3.2+

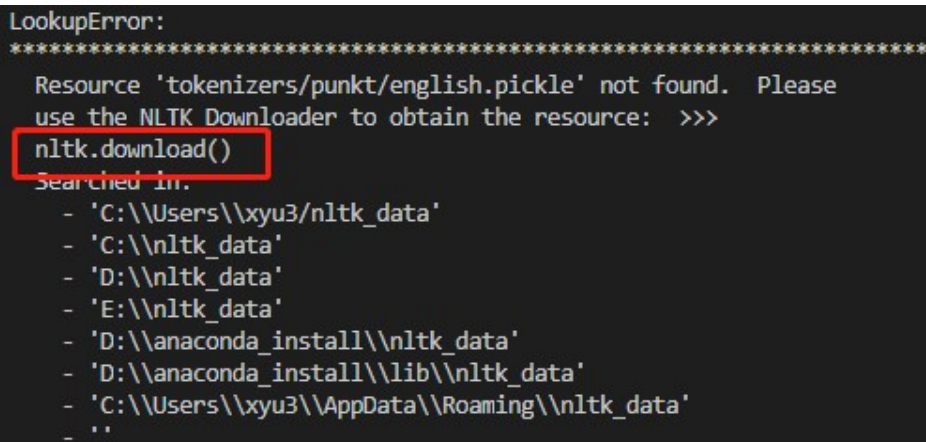
If you install Anaconda, you've already installed the NLTK package. If not, run the command below.

```
pip install nltk
```

Check if you've installed NLTK library.

First, type `python` to go into the py env. Then type the command below, and see if there is any error message.

```
>>> from nltk.corpus import stopwords
```



```
LookupError:
*****
Resource 'tokenizers/punkt/english.pickle' not found. Please
use the NLTK Downloader to obtain the resource: >>>
nltk.download()
Searched in:
- 'C:\\Users\\xyu3\\nltk_data'
- 'C:\\nltk_data'
- 'D:\\nltk_data'
- 'E:\\nltk_data'
- 'D:\\anaconda_install\\nltk_data'
- 'D:\\anaconda_install\\lib\\nltk_data'
- 'C:\\Users\\xyu3\\AppData\\Roaming\\nltk_data'
- ''
```

When you see error messages like above, just follow the instructions, because you need to install some other libraries to make it work.

Dataset Description

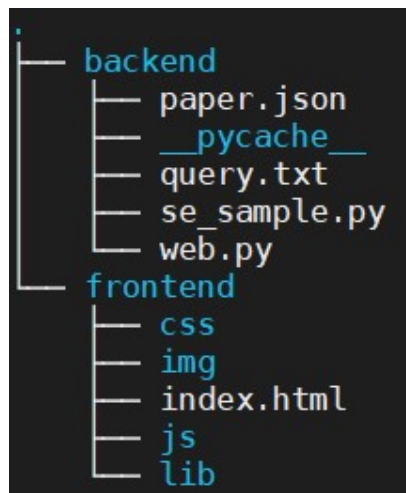
This dataset consists of **778** papers coming from the accepted papers of [ACL 2000](#).

Here is an example for loading the dataset:

```
import json
with open('./paper.json','r') as f:
    papers = json.load(f)
```

```
# papers is a list of dict. Each dict is a record including
title,author,abstract,publish year.
{
    "title": "Low-Resource Generation of Multi-hop Reasoning Questions",
    "authors": ["Jianxing Yu", "Wei Liu", "Shuang Qiu", "Qinliang Su", "Kai
wang", "Xiaojun Quan", "Jian Yin"],
    "abstract": "This paper focuses on generating multi-hop reasoning questions
from the raw text in a low resource circumstance. Such questions have to be
syntactically valid and need to logically correlate with the answers by deducing
over multiple relations on several sentences in the text. Specifically, we first
build a multi-hop generation model and guide it to satisfy the logical
rationality by the reasoning chain extracted from a given text. Since the
labeled data is limited and insufficient for training, we propose to learn the
model with the help of a large scale of unlabeled data that is much easier to
obtain. Such data contains rich expressive forms of the questions with
structural patterns on syntax and semantics. These patterns can be estimated by
the neural hidden semi-Markov model using latent variables. With latent patterns
as a prior, we can regularize the generation model and produce the optimal
results. Experimental results on the HotpotQA data set demonstrate the
effectiveness of our model. Moreover, we apply the generated results to the task
of machine reading comprehension and achieve significant performance
improvements.",
    "year": "2020"
}
```

We will give you a folder called flask_se. The structure is shown below. You will **find the dataset (paper.json) inside the backend folder**. Also, the query file is under the same directory.



Tasks

The basic task is to implement a search engine that supports **abstract searching**.

a) Preprocess the abstract with NLTK

1. Discard punctuation marks
2. Perform tokenization, stop words removal and stemming

b) Create an index (inverted file) of the preprocessed documents. Each postings (an entry in a postings list) contains the document id, and the positions of the indexed word in the documents. You can use the json format to store the inverted index. More specifically, you can use **dict** to store the information.

```
{
  "live":{
    # 40 is the document id which contains the word live and as for 2,5 is
    the position of the word live that appears in the abstract.
    40:[2,5],
    56:[10]
  },
  "only":{
    20:[30],
  }
}
```

Build auxiliary data structures, e.g., word -> document-frequency table. This is helpful when you need to calculate the idf values which need the df (document frequency) values.

c) For each query in the query file, retrieve the top-5 documents using $tf \cdot idf$ weighting and cosine similarity. For each of the top five results, display

1. The document id
2. **Five highest weighted keywords** of the document and the corresponding postings lists
3. The number of unique keywords in the document
4. The magnitude (L2 norm) of the document vector
5. The similarity score

An example is shown below

```
DID
First 20 words of the document
live   -> | D2:1,5 | D3:0 | D6:2 |
never  -> | D5:1 |
only   -> | D6:1 |
tomorrow -> | D1:2 | D2:2 |
twice  -> | D1:0,4 | D2:0,4 |

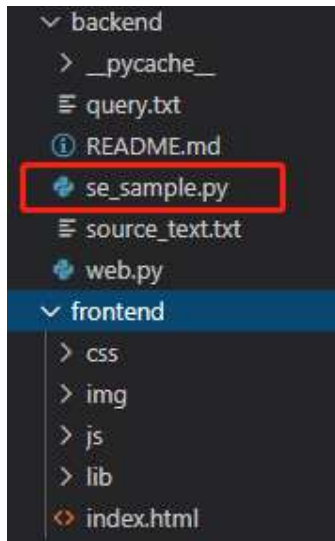
Number of unique keywords in document: ...
Magnitude of the document vector (L2 norm): ...
Similarity score: ...
```

d) In addition to the inverted index. Describe the other data structures you need to support search and ranking.

e) If the text passages are not updated, how would you design your program to speed up the computation of the similarity values?

f) Use Flask to implement a complete search engine application:

- move your py file into the folder "backend" and name it as se_sample.py



- Make sure your se_sample.py file has a function called search_api, and the param is query. The function returns the corresponding texts sorted by the matching score.

```
def search_api(query):  
    '''  
    param: query , str  
    return [] , list of dict, each dict is a paper record of the original  
    dataset  
    '''
```

- install flask
 - if you have not installed the Flask before , try the command `pip install flask`
- Run backend
 - under the path of "backend " , type `python web.py`

```
(root) C:\Users\xyu3\Desktop\flask_se\backend>python web.py  
* Restarting with stat  
* Debugger is active!  
* Debugger PIN: 127-116-582  
* Running on http://localhost:4000/ (Press CTRL+C to quit)
```

- When the server is on, go to the frontend folder and just open the `index.html` with your browser.

Search Engine

type search word

search

- Type the query like "knowledge graph", then click search button

Search Engine

knowledge graph

search

Orthogonal Relation Transforms with Graph Context Modeling for Knowledge Graph Embedding Yun Tang,Jing

Huang,Guangtao Wang,Xiaodong He,Bowen Zhou 2020

Distance-based knowledge graph embeddings have shown substantial improvement on the knowledge graph link prediction task, from TransE to the latest state-of-the-art RotatE. However, complex relations such as N-to-1, 1-to-N and N-to-N still remain challenging to predict. In this work, we propose a novel distance-based approach for knowledge graph link prediction. First, we extend the RotatE from 2D complex domain to high dimensional space with orthogonal transforms to model relations. The orthogonal transform embedding for relations keeps the capability for modeling symmetric/anti-symmetric, inverse and compositional relations while achieves better modeling capacity. Second, the graph context is integrated into distance scoring functions directly. Specifically, graph context is explicitly modeled via two directed context representations. Each node embedding in knowledge graph is augmented with two context representations, which are computed from the neighboring outgoing and incoming nodes/edges respectively. The proposed approach improves prediction accuracy on the difficult N-to-1, 1-to-N and N-to-N cases. Our experimental results show that it achieves state-of-the-art results on two common benchmarks FB15k-237 and WNRR-18, especially on FB15k-237 which has many high in-degree nodes.

Generating Informative Conversational Response using Recurrent Knowledge-Interaction and Knowledge-Copy

Xiexiong Lin,WeiYu Jian,Jianshan He,Taifeng Wang,Wei Chu 2020

Knowledge-driven conversation approaches have achieved remarkable research attention recently. However, generating an informative response with multiple relevant knowledge without losing fluency and coherence is still one of the main challenges. To address this issue, this paper proposes a

Bonus

The upper bound of the bonus is 10 points (out of 100). You can suggest additional features to implement. The followings are just examples:

- Do some statistical analysis on the documents using NLTK and plot the charts, using word distributions, ngrams, etc., and explain the impact of the statistics on the search engine
- Use SpaCy to do some additional tasks to enhance the search engine effectiveness (e.g., name-entity recognition, NER)
- Support author searching, title searching, etc.

- Implement alternative algorithms (e.g., with and without NER) and compare their top-N precision, NDCG, etc.

Note:

Describe your new feature(s) and explain why it/they should be given bonus points. You should focus on the design that you can argue to work well when the data scales up.

What to submit

- **A pdf** including
 - Questions c, d, e described above.
 - Any tricks you have made to improve the searching performance and effectiveness.
- **Code**
 - Just submit the code you've implemented by yourself. Do not upload the whole flask_se folder.
 - Make sure your scripts are runnable.
 - Write down the steps about how to run your code.